FASTLANE
Streamlining Transactions for Low Thread Counts

Jons-Tobias Wamhoff
Christof Fetzer
Technische Universität Dresden, Germany

Pascal Felber
Etienne Rivière
Université de Neuchâtel, Switzerland

Gilles Muller
INRIA, France
Motivation

Number of cores

Performance

Faster

Slower

1

x

Many

Best performance

Expected gains from FastLane

STM

FastLane

Sequential

Many

2
General Idea

• 1 master thread
  • Commits transactions without aborting
  • Minimal instrumentation and bookkeeping
• N helper threads
  • Commit transactions only when not in conflict
  • Contribute progress without impairing on the performance of the master
Code Paths

- Dresden TM Compiler
  - Generates multiple code paths for sequential (uninstrumented), FastLane (master & helper) and STM
  - Generic `START` and `COMMIT` calls with internal branch
  - `READ` and `WRITE` are specific to code path and inlined
    - transaction descriptor only accessed if needed
- TinySTM++ TM runtime
  - Dynamically select code path based on core or thread count at `BEGIN`
Data Structures

Counter
odd: owned
even: otherwise

Memory
... Address written
... Address read
... Address read

Dirty array
Timestamp
Timestamp
...

Master thread
isMaster

Helper thread
Start timestamp
Write-set
Read-set
...
Master Thread

1 function START
2     atomic-or (cntr, MSB)                // Master request priority access
3     while (cntr & 0x01) != 0 do         // Wait for committing helpers
4       wait
5     cntr ← (cntr & ¬MSB) + 1         // Only master can write odd cntr
6 function READ(addr)
7       return *addr                     // No instrumentation
8 function WRITE(addr, val)
9       dirty[hash(addr)] ← cntr         // Mark data as modified
10      addr ← val                       // No additional bookkeeping
11 function COMMIT
12     cntr++                            // Only master can write odd cntr
Helper Thread

1  function START
2        start ← cntr & ~1 & ¬MSB  // Get even counter (discard LSB & MSB)
3  function READ(addr)
4     if CONTAINS(write-set, addr) then  // Is address already in write-set?
5         return GET(write-set, addr)  // Return value previously written
6        val ← *addr
7     if dirty [hash(addr)] > start then  // Validate read value
8         ABORT
9        ADD(read-set, addr)  // Add address to read set
10       return val
11  function WRITE(addr, val)
12     if dirty [hash(addr)] > start then  // Validate write address
13         ABORT
14     PUT(write-set, addr, val)  // Add to (or update) write set
Helper Commit

1 function COMMIT₁
2 if EMPTY(write-set) then
3 \[\text{return}\]
4 repeat
5 \[c \leftarrow \text{WaitForEvenCounter}\]
6 until cas\((cntr, c, c+1)\)
7 if \neg \text{VALIDATE} then
8 \[\text{atomic-dec}(cntr)\]
9 \[\text{ABORT}\]
10 \[\text{EmitWriteSet}\]
11 \[\text{atomic-inc}(cntr)\]
3 Commit Variants

**COMMIT 1**
- acquire(c

**VALIDATE**
- abort
- proceed

**COMMIT 2**
- c = awaitEven(c

**VALIDATE**
- abort
- proceed

**COMMIT 3**
- c = awaitEven(c

**VALIDATE**
- abort

**tryAcquire(c
- failed
- proceed

Spear et al.: RingSTM: Scalable Transactions with a Single Atomic Instruction, SPAA ’08
Optimizations

• **High contention** on *cntr*:
  • Master increments only on helper request when
    1. Helper needs to commit
    2. Helper aborts after failed validation
  • **Avoid atomic operations** with commit variant 2 & 3
    • *helpers* lock synchronizes helpers at commit
    • Only 2 thread synchronization (still cache invalidation)
  • **Pre-validate** whenever waiting for even *cntr*
Evaluation intset

Throughput (million tx/second)

Sequential
ETL
WT
fastlane-1
fastlane-2
fastlane-2-opt
fastlane-3

RB 4096 5%
RB 4096 20%
SL 1024 0%
SL 1024 20%
LL 1024 5%
LL 8192 20%
SL 1024 0%
HS 1024 5%
HS 1024 0%
Commit/Abort Rates

Hash Set (5% updates)

Red-Black Tree (5% updates)
Evaluation STAMP

Figures 3, 4, 5, 7

FastLane-1
FastLane-2
FastLane-3

Threads

Time (s)

genome

intruder

kmeans

ssca2

vacation

Sequential
ETL
WT
FastLane-1
FastLane-2
FastLane-3

Commits/Aborts (millions per second)
Irrevocability

- Assign master role dynamically to any thread
  - Default: 1st thread becomes master
  - Acquire `cntr` and set global master flag
  - Save to execute legacy code and system calls without non-transactional data accesses
- Quiescence lock will stop all other threads
  - Acquire `cntr` to stop master and set quiescence flag
  - Wait for all other transactions to finish
  - Used to switch modes
Thank you!
3 Commit Variants

- Must validate accessed memory not updated after start
- Differ in validation and \( cntr \) obtainment strategy
  1. Validate in isolation after acquiring \( cntr \)
  2. Obtain even \( cntr \) and pre-validate, acquire \( cntr \) and re-validate if \( cntr \) changed externally
  3. Obtain even \( cntr \) and validate, try to acquire \( cntr+1 \) or start over
Helper Commit 1

1 function COMMIT₁
2 if EMPTY(write-set) then // Read-only transaction?
3 \[ \text{return} \] // Commit immediately
4 repeat // Try acquiring counter
5 \[ \text{c} \leftarrow \text{WaitForEvenCounter} \]
6 until cas\((cntr, c, c+1)\) // Attempt C&S only after counter seen even
7 if \(\neg\text{VALIDATE}\) then
8 \[ \text{atomic-dec}(cntr) \] // Release counter upon failed validation
9 \[ \text{ABORT} \]
10 \[ \text{EmitWriteSet} \] // Write updates to memory
11 \[ \text{atomic-inc}(cntr) \] // Release counter
Helper Commit 2

1 function COMMIT\_2
2 \hspace{1em} \textbf{if} EMPTY(write-set) \textbf{then} \hspace{1em} \text{\ } // Read-only transaction?
3 \hspace{3em} \textbf{return} \hspace{1em} \text{\ } // Commit immediately
4 \hspace{1em} \textbf{lock} (helpers) \hspace{1em} \text{\ } // Only one helper at a time (FIFO lock)
5 \hspace{1em} c \leftarrow \text{WAITFOREVENCOUNTER}
6 \hspace{1em} \textbf{if} \neg \text{VALIDATE} \textbf{then} \hspace{1em} \text{\ } // Pre-validate before acquiring counter
7 \hspace{3em} \textbf{unlock} (helpers) \hspace{1em} \text{\ } // Release lock upon failed validation
8 \hspace{3em} \text{ABORT}
9 \hspace{1em} t \leftarrow c+1 \hspace{1em} \text{\ } // Remember validation time
10 \hspace{1em} \textbf{while} \neg \text{cas}(cntr, c, c+1) \textbf{do} \hspace{1em} // Likely commit: try acquiring counter
11 \hspace{3em} c \leftarrow \text{WAITFOREVENCOUNTER}
12 \hspace{1em} \textbf{if} cntr > t \land \neg \text{VALIDATE} \textbf{then} \hspace{1em} // Check that validation still holds
13 \hspace{3em} \text{atomic-dec}(cntr) \hspace{1em} // Release locks upon failed validation
14 \hspace{3em} \textbf{return} \hspace{1em} \text{ABORT}
15 \hspace{1em} \text{EmitWriteSet} \hspace{1em} \text{\ } // Write updates to memory
16 \hspace{1em} \text{atomic-inc}(cntr) \hspace{1em} \text{\ } // Release locks
17 \hspace{1em} \textbf{unlock} (helpers)
Helper Commit 3

1 function COMMIT₃
2 if EMPTY(write-set) then // Read-only transaction?
3 \quad return // Commit immediately
4 lock (helpers) // Only one helper at a time (FIFO lock)
5 repeat
6 \quad c \leftarrow \text{WaitForEvenCounter}
7 \quad if \neg\text{VALIDATE} then // Pre-validate before acquiring counter
8 \quad \quad unlock (helpers) // Release lock upon failed validation
9 \quad \quad ABORT
10 until c = cntr \wedge \text{cas}(cntr, c, c+1)
11 \text{EmitWriteSet} // Write updates to memory
12 \text{atomic-inc}(cntr) // Release locks
13 unlock (helpers)